UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

SELF-CONFIGURING SOCIO-TECHNICAL SYSTEMS: REDESIGN AT RUNTIME

Volha Bryl and Paolo Giorgini

July 2006

Technical Report # DIT-06-048

Self-Configuring Socio-Technical Systems: Redesign at Runtime

Volha Bryl Paolo Giorgini

Department of Information and Communication Technology,
University of Trento, Italy
Email: {bryl, paolo.giorginig}@dit.unitn.it

Abstract: Modern information systems are becoming more and more socio-technical systems, namely systems composed of human (social) agents and software (technical) systems operating together in a common environment. The structure of such systems has to evolve dynamically in response to the changes of the environment. When new requirements are introduced, when an actor leaves the system or when a new actor comes, the socio-technical structure needs to be redesigned and revised. In this paper, an approach to dynamic reconfiguration of a socio-technical system structure in response to internal or external changes is proposed. The approach is based on planning techniques for generating possible alternative configurations, and local strategies for their evaluation. The reconfiguration mechanism is presented, which makes the socio-technical system self-configuring, and the approach is discussed and analyzed on a simple case study.

Keywords: self-configuration, socio-technical systems, planning, local strategies

1. Introduction

Socio-technical systems (STS), as opposed to the traditional technical computer-based systems, include human agents as an integral part of their structure. One important aspect of an STS is its dynamicity: an STS operates in continuously changing environments and, accordingly, its structure changes dynamically. Unlike the technical computer-based systems, an STS includes the knowledge of how the system should be used to achieve some organizational objectives, and is normally regulated and constrained by internal organizational rules, external laws and regulations [16]. So for example, a conference reviewing system, which consists of both human agents and software components, has to conform to the rules of the reviewing process. All this calls for the new type of requirements that introduce the need of highly adaptable and reconfigurable systems, see e.g. [17].

Recently, a lot of work has been devoted to the problem of dynamic reconfiguration and adaptation of software systems [10, 8, 4, 12, 9, 18]. Attempts to adjust the existing agent-oriented methodologies, such as Gaia [8], or to create a specialized ones, such as Adelfe [4], to develop adaptive agents are described in the literature. All these proposals can be grouped in approaches that consider the reconfiguration process from the local and from the global perspective. Self-configuration from the local perspective, i.e. on the level of an individual agent, is related to the concept of self-organization. Self-organization phenomena (see e.g. [20]) is observed when some macroscopic system properties arise (emerge) dynamically from the local micro-level interactions. However, such perspective is sometimes not enough as it does not allow to reach all the desired properties of an STS which works in the dynamic environment [10]. For example, the social behavior of being helpful, or following the imposed external laws, is difficult to describe by the "individual rationality" principle assumed by self-organization emergent models. Another example is a scientific institution, which could hardly function on the base of self-organization principles, without any centralized authority. Differently, this paper follows the approach presented in [10], which suggests combining the perspective of individual agents with the global one, in which reconfiguration is controlled centrally.

In this paper an approach to the problem of dynamic reconfiguration of an STS structure in response to the internal and/or environmental changes is proposed. The approach is based on planning techniques (used to explore the space of alternative configurations), combined with evaluating the generated alternative in terms of local strategies of the system actors. The approach comprises the following steps.

- Identify system actors, their goals, capabilities, and interrelations.
- Select the initial configuration by the following three-step iterative procedure: (i) construct the assignment of goals to actors with the help of planning, so that all goals are to be satisfied; (ii) evaluate the obtained assignment with respect to local interests of the system actors, in order to identify which actors will be motivated to deviate from the assignment; (iii) in case the deviation is inevitable, reformulate the planning problem, and go to the construction of the next assignment.
- Monitor the STS and the environment, in case of changes assess whether the reconfiguration is necessary. Reconfigure the system with the help of the above described iterative procedure.

There exists a number of works which are, to some extent, similar to the approach presented in this paper. [10] deals with the problem of dynamic reorganization of agent societies, and presents the classification of reorganization situations. According to the authors, the paper is exploratory in nature, and contains the discussion of the problem rather then the possible solutions. In [12] *Moise+* organizational model is extended to support the reorganization of multi-agent systems. The organization is represented along its structural and functional dimensions, and the deontic relation among these dimensions is defined. The reorganization

process is performed by the set of organization agents playing the specific roles, such as *OrgManager* that is in charge of managing the reorganization process, *Monitor* that is monitoring the system, *Designer* that develops reorganization proposals, and the like. However, no specific guidelines are provided on how the new system configuration should be constructed. [9] describes how to design adaptive multi-agent systems using the organizational model that consists of a structural and state models of an organization, and a transition function from one organizational state to another. The structural model contains the information about goals, agent roles, organizational rules and laws. A state model is an instance of an organization which includes a set of agents together with the relationships between them and other structural model components. A number of events called reorganization triggers are described, which may cause the system reorganization. The reorganization process is assumed to be application specific, and the selection of an appropriate configuration relies on maximizing a sort of utility function, so called organization capability score. In [18] the techniques for organization and reorganization of multi-agent systems in the domain of oceanography are presented. The reorganization is domain specific, and is based on communication protocols, with the help of which groups of agents cooperate and reorganize themselves in response to the environmental changes.

The approach proposed in this paper differs from the above described work as it provides, independently of the domain, the concrete guidelines on how the reorganization process could be organized. Another important point is the automation support – the process of exploring the space of alternative system configurations is performed automatically with the help of planning techniques. Also the local strategies of an STS actors are taken into account, which allows each actor to evaluate the new configuration from the local perspective, and deviate from it if the load/risk/complexity of the new assignment is unacceptable for this actor. Taking the local strategies of the system actors into consideration makes our approach particularly useful for the socio-technical systems, because strategic and rational behavior is intrinsic to the human actors.

The rest of the paper is structured as follows. In Section 2 an illustrative example is presented, and then the procedure of generation and evaluation of alternative configurations is described, and it is shown how this procedure works with the help of the above mentioned example. Then, in Section 3 the reconfiguration mechanism is introduced, again illustrated with the example. In Section 4 implementation issues are discussed, which is followed by the conclusive remarks in Section 5.

2. Approach

In this section first the example is introduced, which will be used through all the paper. Then it is explained how to use planning techniques for generating alternative configurations, and finally the procedure for their evaluation is presented.

2.1 Example

The example presented in this section was selected to be quite simple for the reasons of compactness and ease of understanding.

Consider a small firm which sells office equipment to its customers. The office equipment is supplied by two companies, *MediaMarket* and *HWStore*, both having a database containing information about supplied goods, their technical characteristics and prices. To organize the placing of orders for the sell items, the supporting software system of the firm comprises a subcomponent, called *search-and-order multi-agent system (MAS)*, which consists of three agents, see Fig. 1. These agents can process the user orders, i.e. search for the required item in the supplier's database, provide information to the customer if the item was found, and formulate the request to the supplier otherwise. Two of these agents, A_{MM} and A_{HWS}, can work only with the database of one supplier, *MediaMarket* and *HWStore*, respectively. In other words, because of the (limited) capabilities A_{MM} possesses, it cannot work with *HWStore* database, and vice versa. The third agent A_{UNIV} is a reserve one, it can query both databases, although, less efficiently than A_{MM} and A_{HWS}, and is used when other agents are unable to hold numerous requests of a user (a clerk of a firm).

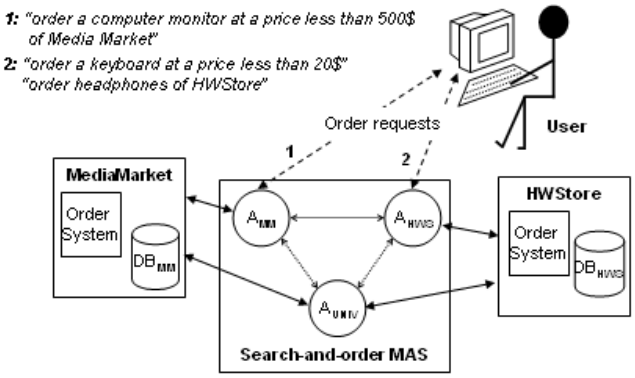

Fig. 1. Search-and-order MAS.

In principle, A_{UNIV} can be a human agent which is exploited only if some critical situation occurs: satisfying the customer's request needs some specific human support (e.g. making a phone call), or the software component fails and the customer remains unserved, which violates the organizational rules, etc. However, as the approach proposed in this paper treats both artificial and human actors in a similar way, human actors will not be introduced in the example due to the space and simplicity reasons. Also note that the number of suppliers is limited to two just for the sake of simplicity. In reality in such a system there can be tens, or even hundreds of different suppliers, and a limited number of agents, each with different set of capabilities. Some of these agents are more efficient when working with one specific "type" of suppliers, while others are "universal", i.e. can work with all suppliers. The task of allocating the incoming orders in a (sub)optimal way is indeed challenging – and this is what is going to be automated with the help of the approach proposed in this paper.

Suppose that initially, i.e. at time point t_0, there are three requests the agents have to satisfy. One query – "*order a computer monitor at a price less than 500$ of MediaMarket*"

– is sent by a user to A_{MM}, and the two – "*order a keyboard at a price less than 20$*" and "*order headphones of HWStore"* – are sent to A_{HWS}. Even for this simple example there are a number of alternative initial configurations. E.g. the query "*order a keyboard at a price less than 20$"* could be accomplished by searching either in the database of *MediaMarket* or *HWStore*. Another source of alternatives is whether to involve A_{UNIV} in performing the queries or not. Thus, the problem is how to assign queries to agents in a (sub)optimal way.

2.2 Generating alternative configurations

In our approach the configuration of an STS is described in terms of dependencies among actors for goals. In this the approach is following such frameworks as *i** [21] and Tropos [5], where the functional requirements to the system are conceived as networks of delegation dependencies. Every delegation involves two actors, where one actor delegates to the other the fulfillment of a goal. The later actor, called delegatee, can either fulfill the delegated goal, or further delegate it, thus creating another delegation relation in the network. In this paper it is proposed to frame the task of constructing such networks as a *planning problem*: selecting a suitable configuration corresponds to selecting a plan that satisfies the goals of human and software actors.

The basic idea behind planning approach is to automatically determine the course of actions (i.e. a plan) needed to achieve a certain goal where an action is a transition rule from one state of the system to another [19, 15]. Planning is useful in the situations where it is not feasible to enumerate in advance the possible transitions from the initial to the desired state [3]. Actions are described in terms of preconditions and effects: if a precondition of an action is true in the current state of the system, then the action is performed. As a consequence of an action, the system will be in a new state where the effect of the action is true. A specification language is required to represent the planning domain, i.e. the initial and the desired states of the system, and the actions. Once the domain is described, the solution to the planning problem is the (not necessarily optimal) sequence of actions that allows the system to reach the desired state from the initial state.

In Table 1 predicates used to describe the planning domain are introduced. Predicates take variables of three types: actors, goals and goal types. To typify goals, *type* predicate is used. Actor capabilities are described with *can_satisfy_gt* predicate, which means that an actor has enough capabilities to satisfy any goal of a specific type. Social dependencies among actors are reflected by *can_depend_on* predicate, which means that one actor can delegate to another actor the fulfilment of any goal. Predefined ways of goal refinement are represented using *and/or_decomposition* predicates. The initial desires of actors are represented with *wants* predicate. When a goal is fulfilled, *satisfied* predicate becomes true for it.

A plan, which is constructed to fulfill the goals of the system actors, comprises the following actions.

- *Goal satisfaction.* An actor can satisfy a goal only if the achievement of this goal is among its desires and it actually possesses the capabilities to satisfy it.

Table 1: Predicates.

Goal Properties
type(g : goal, gt : gtype)
and_decomposition$_n$(g : goal, g_1 : goal, …, g_n : goal)
or_decomposition$_n$(g : goal, g_1 : goal, …, g_n : goal)
satisfied(g : goal)
Actor Properties
can_satisfy_gt(a : actor, gt : gtype)
can_depend_on(a : actor, b : actor)
wants(a : actor, g : goal)

- *Goal delegation*. An actor may have not enough capabilities to achieve its goals by itself, and so it has to delegate their satisfaction to other actors. The decision on how to satisfy a goal – by its own capabilities or by further delegation – is up to the delegatee.
- *Goal decomposition/refinement.* Two types of goal refinement are supported: OR-decomposition, which suggests the list of alternative ways to satisfy a goal, and AND-decomposition, which refines a goal into subgoals which all are to be satisfied in order to satisfy the initial goal.

When the problem domain and the problem itself are formally represented, a *planner* is used to produce a solution. An important point of the approach is that it is not intended to invent a special-purpose planning tool – instead, the idea is to choose a suitable one among the available off-the-shelf planners [15]. After analyzing a number of planners (see [7] for the details), LPG-td [13] have been chosen for implementing the planning domain, which is a fully automated system for solving planning problems, which supports PDDL 2.2 specification (Planning Domain Definition Language) [11].

Let us illustrate how the planning part of the approach works with the example described in Section 2.1.

The question is what the initial configuration of the search-and-order MAS is. The formalization of this problem, in terms of the predicates introduced above, is presented in Fig. 2.

type (OrderMonitorOfMM, tDB1)
type (OrderKeyboardOfMM, tDB1)
type (OrderKeyboardOfHWS, tDB2)
type (OrderHeadphonesOfHWS, tDB2)
can_depend (A_{MM}, A_{HWS}) can_depend (A_{HWS}, A_{MM})
can_depend (A_{MM}, A_{UNIV}) can_depend (A_{UNIV}, A_{MM})
can_depend (A_{HWS}, A_{UNIV}) can_depend (A_{UNIV}, A_{HWS})
can_satisfy_gt (A_{MM}, tDB1)
can_satisfy_gt (A_{HWS}, tDB2)
can_satisfy_gt (A_{UNIV}, tDB1)
can_satisfy_gt (A_{UNIV}, tDB2)
or_decomposition$_2$ (OrderKeyboard,
    OrderKeyboardOfMM, OrderKeyboardOfHWS)
wants (A_{MM}, OrderMonitorOfMM)
wants (A_{HWS}, OrderHeadphonesOfHWS)
wants (A_{HWS}, OrderKeyboard)

Fig. 2. Planning problem formalization.

The plan P_0 generated by the planner is presented in Fig. 3. Note, that there are several alternative configurations in which all goals are satisfied, e.g. the goal *OrderKeyboard* can

be achieved by A_{MM} by satisfying the *OrderKeyboardOfMM* or-subgoal, instead of *OrderKeyboardOfHWS*; or the goal *OrderHeadphonesOfHWS* can be delegated to A_{UNIV} and satisfied by it. The configuration presented in Fig. 3 is just the *first* one generated by the planner, which means that it was the first plan (e.g. the shortest one) the planner found to satisfy all the goals. The idea is to take this first solution as a starting point, evaluate it with respect to the individual interests of each system actor (see Section 2.3 for the further explanations), and ask the planner for the next solution only if the current one appears to be unsatisfactory.

(SATISFIES A_{MM} OrderMonitorOfMM)
(SATISFIES A_{HWS} OrderHeadphonesOfHWS)
(OR_DECOMPOSES A_{HWS}, OrderKeyboard
OrderKeyboardOfMM OrderKeyboardOfHWS)
(SATISFIES A_{HWS} OrderKeyboardOfHWS)

Fig. 3. Initial configuration.

The graphical representation of the obtained configuration is depicted in Fig. 4. Circles represent agents, with their goals represented as ovals, goals with underlined description text in the balloon of an agent means that these goals are to be satisfied by this agent.

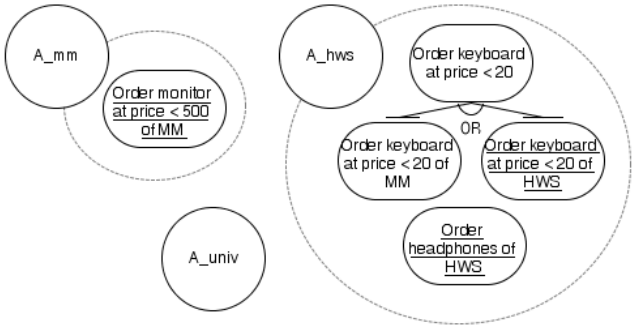

Fig. 4. Initial configuration of search-and-order MAS.

The approach described in this subsection was applied to the design of secure systems [7].

2.3 Evaluating alternative configurations

After an alternative STS configuration has been generated by the planner, it must be evaluated to check that it is not in conflict with the individual interests of system actors. Actors of a socio-technical system can be seen as players in a game-theoretic sense as they are self-interested and rational. This might, for example, mean that they want to minimize the load imposed personally on them, i.e. to reduce the number and the complexity of actions they are involved in. Ideas from game theory [14] can be used to determine whether an alternative satisfies all the system actors. Another way to say it in game theoretic terms is "whether an alternative is an equilibrium". In particular, an alternative is an equilibrium if no actor can do better with respect to its own goals by adopting a different strategy for delegating goals/accepting delegations/refining goals/etc. Within the framework, the following evaluation schema is used.

First, for all actors a_i, $i=1..n$ and all goals g_k, $k=1..m$, where n and m are the number of actors and goals, respectively, the costs are defined:

- cs_{ik} is the cost for the actor a_i of satisfying the goal g_k;
- cr_{ik} is the cost for the actor a_i of decomposing (refining) the goal g_k;
- cd_{ijk} is the cost for the actor a_i of delegating a goal g_k to the actor a_j.

Costs are specific to a given STS and may reflect the complexity of an action, the risks an actor could face while/after the action is performed, etc. It is assumed that the costs are explicitly given as an input, and do not change while the system is functioning. Also note, that in this work the focus is on the load constraints only, and not on the other factors which may influence the player's decision to deviate, e.g. risk concerns.

Then, the cost of a given alternative P for the actor a_i is calculated by summing up the costs of actions in P which a_i is involved in, and is denoted by

$$c(a_i, P) = \sum_{delegate(a_i, a_j, g_k) \in P} cd_{ijk} + \sum_{decompose_l(a_i, g_k, g_{k1}, ..., g_{kl}) \in P} cr_{ik} + \sum_{satisfy(a_i, g_k) \in P} cs_{ik} \quad (1)$$

where $decompose_l(a_i, g_k, g_{k1}, ..., g_{kl})$ stands for the decomposition of g_k into l subgoals $g_{k1}, ..., g_{kl}$.

After the costs are computed, for each actor the conditions are defined upon which an actor decides whether to deviate from an alternative P or not. In particular, the actor a_i whose predefined upper cost bound c_i^{up} is greater than $c(a_i, P)$ will be willing to deviate from P.

The evaluation procedure itself is the following.

- An alternative P is generated with the help of a planner.
- Cost $c(a_i, P)$ is calculated for each actor a_i.
- Actor a_{min} is identified whose value of $c(a_{min}, P)$ is minimal among all actors who want to deviate from P.
- The first most expensive action d_{worst} (an action with the highest cost) is identified among the actions of P in which a_{min} is involved.
- Negation of d_{worst} is added to the initial planning problem, and replanning is performed. If no plan can be found, the next d_{worst} (an action with the next highest cost) is identified.

Ideally, the process stops when an equilibruium-like solution is found, i.e. no actors are willing to deviate from it. If such a solution is impossible to find then the existing constraints might be relaxed: load bounds decreased, or even actors' goals and capabilities revised. This problem will likely need the interference of a human designer, and is not addressed in this paper.

Let us come back to the search-and-order MAS example. It is assumed that different order queries have different costs for the three system agents, depending, e.g., on the complexity of a query. The costs for A_{UNIV} are higher, which is caused by its "universality", i.e. the ability to work with both suppliers. Moreover, the order queries are subdivided into two classes – "simple" and "complicated". The cost of the satisfaction of a simple query is lower than a corresponding cost for the complicated query. An example of a complicated query could be "*order best-selling HDD of HWStore*", as it requires obtaining the statistical information.

The cost of any delegation is equal to 1 unit, the cost of a decomposition is equal to 2 units. For A_{MM} and A_{HWS} performing simple order queries costs 10 units, performing complicated ones – 15 units, for A_{UNIV} – 15 and 20 units, respectively. Tolerable bound of load for all three agents, under which they are not willing to deviate from the imposed reconfiguration plan, is equal to 30 units. In these conditions the costs of the obtained initial configuration plan P_0 (see Section 2.2) are the following: $c(A_{MM}, P_0)=10$, $c(A_{HWS}, P_0)=2+10+10=22$, and $c(A_{UNIV}, P_0)=0$. Due to simplicity of the example, this first solution generated by the planner is satisfactory from the point of view of all three agents, i.e. the evaluation shows that the plan costs are within the tolerable load bounds for each agent.

3. Redesign at Runtime

In this section a centralized reconfiguration mechanism is introduced, which is based on the planning-and-evaluation approach proposed in the previous section. The reconfiguration steps described below can be performed by one special-purpose system actor, or a group of existing system actors. Some implementation issues will be overviewed in Section 4.

3.1 Reconfiguration Mechanism

The reconfiguration mechanism

- collects and manages the information about the system;
- evaluates the load imposed on each system actor based on the local utilities of the actors to decide whether the system needs to be redesigned in response to external or internal changes;
- and, if the above evaluation shows that the reconfiguration is needed, replans the system structure in order to optimize the distribution of load imposed on system actors.

It stores and updates

- the current problem definition *problemDef*, i.e. *actor* and *goal* variables, and predicates (see Table 1) describing them;
- the list of all goals $G=\{g_i, i=1..n\}$ present in the system together with their states (described in the next paragraph), also for each goal the actor who initially wanted it to be satisfied is stored;
- the current plan of actions, i.e. a list of actions $D=\{d_j, j=1..m\}$ generated during the last (re)design iteration and not accomplished so far;
- archived data, e.g. *actionLog*.

To describe the states of the goals in G, the two of already introduced predicates are used, namely, *wants(a : actor, g : goal)* and *satisfied(g : goal)*. In addition, a predicate *committed(a : actor, g : goal)* is introduced. Predicate *committed(a, g)* becomes true when a reconfiguration mechanism is notified that a has committed to g, meaning that a has taken a decision to satisfy g. This predicate is used to support the *minimal change* principle during the reconfiguration process. As it will be seen from the algorithm presented in this section, the reconfiguration does not apply to the *committed* goals, and thus, not all the STS structure is revised each time. Note that *satisfied(g)* implies *not committed(a, g)*.

The reconfiguration algorithm is presented in Fig. 5, and is organized in a way that a block corresponds to one internal or environmental change. The notification about the change is obtained either from the inside of the system or from the environment. Each system actor is obliged to communicate to some central point if it has committed to, or achieved a goal. In order to avoid continuous replanning, a time slot φ is introduced, such that triggering events initiate evaluation and replanning only if the time passed since the last replanning is greater than φ **(line 0)**.

In the following each block is explained briefly.

- **(lines 1–3)** An actor a has committed to do a goal g. In this case *committed(a,g)* is set to be true, and all decompositions and delegations of g are moved to the action log.
- **(lines 4–8)** An actor a has achieved a goal g. In this case *satisfied(g)* is set to be true, and satisfaction action is moved from D to the action log. Then it is evaluated whether the actor that has satisfied the goal is "free enough", in a sense whether the total cost of the actions in D it is involved in is less than a predefine threshold. If it is the case, the replanning with non-committed goals procedure *ReplanWithG*, presented in Fig. 6 and described below, is performed.
- **(lines 9–17)** One of the imposed requirements is relaxed, i.e. a goal g is no longer needed to be achieved. It is assumed that g is not a subgoal of any other goal present in the system. In this case the corresponding variable *g : goal* and predicates in which g appears are removed from the problem definition. All action containing g are moved from D to the removed action log. The same "removal procedure" is done for each subgoal of g. Then it is evaluated whether any of the system actors is "free enough" in the above defined sense, and if such actors exist, the *ReplanWithG* replanning procedure is performed.
- **(lines 18–24)** One of the existing actors leaves the system. For each goal that was initially wanted by this actor, the above described "removal procedure" is performed. Then the corresponding variable *a : actor* and predicates in which a appears are removed from the problem definition, all actions containing a are moved from D to the action log. All goals which was wanted by a, or which a was committed to, are considered to be new to the system, and the general replanning procedure *Replan*, presented in Fig. 6 and described below, is performed.
- **(lines 25–27)** A new actor joins the system. In this case a new variable *a : actor* appears in the problem definition together with the predicates describing the properties of a. Then the *ReplanWithG* replanning procedure is performed.
- **(lines 28–30)** A new requirement to the system has been introduced, i.e. a new goal g is now to be satisfied. In this case a new variable *g : goal* appears in the problem definition together with the predicates describing the properties of g, and for some actor a of the system *wants(a, g)* becomes true. Then the *Replan* replanning procedure is performed.

In Fig. 6 the replanning procedures used through the algorithm are introduced.

```
0   if notified(notifMess) and t_curr − t_prev ≤ φ quit
1   if notifMess = "a has committed to g" then
2         G : committed (a, g)← true
3         move Decompose(.., g, ...) and Delegate(.., g, ..) from D to ActionLog
4   if notifMess = "a has achieved g" then
5         G : committed (a, g)← false, satisfied (g)← true
6         move Satisfy(a, g) from D to ActionLog
7         if load(a) ≤ freeEnoughBound then
8               ReplanWithG(∅)   /* with empty set of new goals */
9   if notifMess = "g is removed" then
10        if g is a subgoal of a valid goal then quit   /* abnormal situation */
11        remove variable g and predicates containing g from problemDef
12        move actions containing g from D to removedActionLog
13        for each subgoal g_sub of g, s.t. g_sub is not a (subgoal of any) valid goal
14              repeat lines 11–12
15        for each actor a if load(a) ≤ freeEnoughBound then
16              ReplanWithG(∅)   /* with empty set of new goals */
17              exit for each loop
18  if notifMess = "a leaves the system" then
19        for each g initially wanted by a   /* information stored in G */
20              repeat lines 11–12
21        remove variable a and predicates containing a from problemDef
22        move actions containing a from D to removedActionLog
23        remove predicates containing a from G, put affected goals to newGoalsList
24        Replan(newGoalsList)
25  if notifMess = "new a introduced" then
26        add a variable and related predicates to problemDef and G
27        ReplanWithG(∅)   /* with empty set of new goals */
28  if notifMess = new g introduced" then
29        add g variable and related predicates to problemDef and G
30        Replan({g})
```

Fig. 5. Reconfiguration algorithm.

```
ReplanWithG(newGoals)
    construct workingProblemDef :
        problemDef with a problemGoal formed as a conjunction of satisfied(g),
            s.t. g ∈ newGoals or g ∈ G and ¬commited(·, g) is true
    iteratively
        plan for workingProblemDef to get newPlan
        evaluate newPlan together with
            Satisfy(·, g) ∈ D s.t. commited(·, g) is true
    if newPlan is satisfactory then
        replace D with newPlan together with
            Satisfy(·, g) ∈ D s.t. commited(·, g) is true

Replan(newGoals)
    construct workingProblemDef :
        problemDef with a problemGoal formed as a conjunction of satisfied(g),
            s.t. g ∈ newGoals
    iteratively
        plan for workingProblemDef to get newPlan
        evaluate newPlan together with D
    if newPlan is satisfactory then
        append newPlan to D
    else ReplanWithG(newGoals)
```

Fig. 6. Replanning procedures.

- **ReplanWithG**. First planning for all new goals and goals in *G*, except for committed ones, is performed; then the evaluation is done for the intermediate plan – the new plan in which additional actions are included, *Satisfies(a,g)* ∈ *D*, such that *commited(a,g)* is true. If the evaluation is successful, *D* is replaced with the intermediate plan.
- **Replan**. Planning is performed for the new goals, and then the intermediate plan – *D* plus the new plan – is evaluated. If the evaluation is successful, new actions are added to *D*. If not the **ReplanWithG** is performed.

If it is still impossible to find a plan of actions to satisfy the system goals, then the commitments of actors to goals might be revised. However, this feature is not yet supported by the framework, and is not addressed in this paper.

3.2 Example: reconfiguration process

Let us now illustrate the proposed procedure with the help of the example presented in Section 2.1, and discussed in the previous section. Here the reaction of the reconfiguration mechanism to the two triggering events, occurred at the time steps t_1 and t_k, is considered.

Step t_1. Suppose that a new request has arrived to the agent A_{HWS}, *"order speakers at price between 10 and 30$ of HWStore"*, which is classified as simple. Till that moment A_{HWS} has committed to *"order headphones of HWStore"*.

Replanning only for the new goal gives the resulting plan P_1 presented in Fig. 7.

(SATISFIES A_{MM} OrderMonitorOfMM)
(SATISFIES A_{HWS} OrderHeadphonesOfHWS)
(OR_DECOMPOSES A_{HWS}, OrderKeyboard
OrderKeyboardOfMM OrderKeyboardOfHWS)
(SATISFIES A_{HWS} OrderKeyboardOfHWS)
(SATISFIES A_{HWS} OrderSpeakersOfHWS)

Fig. 7. First plan at time step t_1.

The costs for the obtained plan P_1 are the following: $c(A_{MM}, P_1)=10$, $c(A_{HWS}, P_1)=2+10+10=32>30$, and $c(A_{UNIV}, P_1)=0$. As far as A_{HWS} is not satisfied with the imposed load, replanning for all the goals, except committed, is performed.

The resulting plan P_2 is illustrated in Fig. 8.

(SATISFIES A_{MM} OrderMonitorOfMM)
(OR_DECOMPOSES A_{HWS}, OrderKeyboard
OrderKeyboardOfMM OrderKeyboardOfHWS)
(PASSES A_{HWS} A_{MM} OrderKeyboardOfMM)
(SATISFIES A_{MM} OrderKeyboardOfMM)
(SATISFIES A_{HWS} OrderHeadphonesOfHWS)
(SATISFIES A_{HWS} OrderSpeakersOfHWS)

Fig. 8. Second plan at time step t_1.

The costs for the P_2 are the following: $c(A_{MM}, P_2)=10+10=20$, $c(A_{HWS}, P_2)=2+1+10+10=23$, and $c(A_{UNIV}, P_2)=0$. As far as all $c(.,P_2)<30$, the reconfiguration plan P_2 is adopted. The assignment structure is revised, and redesigned as depicted in Fig. 9.

Step t_k. Suppose that a new request has arrived to the agent A_{HWS}, *"order best-selling HDD of HWStore"*, which is classified as complicated. Till this moment A_{MM} has committed to *"order a keyboard at a price less than 20$ of MediaMarket"*, A_{HWS} has committed to *"order headphones of HWStore"*, and A_{UNIV} has committed to a new simple *"goal from A_{MM}"*.

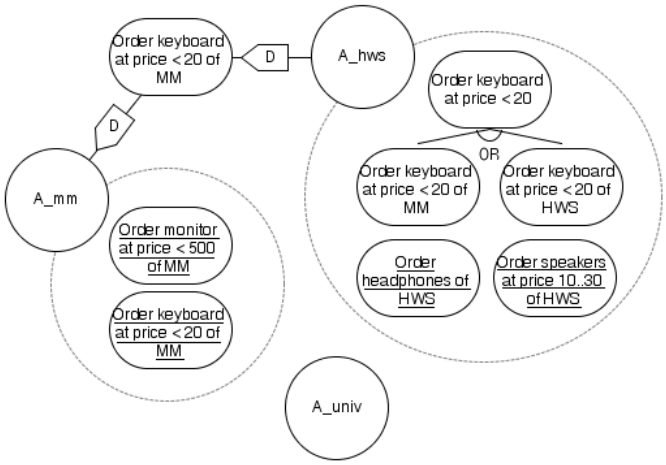

Fig. 9. First reconfiguration of search-and-order MAS.

Replanning only for the new goal gives the resulting plan P_k presented in Fig. 10.

(SATISFIES A_{MM} OrderMonitorOfMM)
(OR_DECOMPOSES A_{HWS}, OrderKeyboard
OrderKeyboardOfMM OrderKeyboardOfHWS)
(PASSES A_{HWS} A_{MM} OrderKeyboardOfMM)
(SATISFIES A_{MM} OrderKeyboardOfMM)
(SATISFIES A_{HWS} OrderHeadphonesOfHWS)
(SATISFIES A_{HWS} OrderHDDOfHWS)
(SATISFIES A_{HWS} OrderSpeakersOfHWS)
(SATISFIES A_{UNIV} GoalFromA_{MM})

Fig. 10. First plan at time step t_k.

The costs for the P_k are the following: $c(A_{MM}, P_k)=21$, $c(A_{HWS}, P_k)=23+15=38>30$, and $c(A_{UNIV}, P_k)=15$. As far as A_{HWS} is not satisfied with the imposed load, replanning for all the goals, except committed, is performed.

The resulting plan P_{k+1} is presented in Fig. 11.

(SATISFIES A_{MM} OrderMonitorOfMM)
(OR_DECOMPOSES A_{HWS}, OrderKeyboard
OrderKeyboardOfMM OrderKeyboardOfHWS)
(PASSES A_{HWS} A_{MM} OrderKeyboardOfMM)
(SATISFIES A_{MM} OrderKeyboardOfMM)
(SATISFIES A_{HWS} OrderHeadphonesOfHWS)
(SATISFIES A_{HWS} OrderSpeakersOfHWS)
(SATISFIES A_{UNIV} GoalFromA_{MM})
(PASSES A_{HWS} A_{UNIV} OrderHDDOfHWS)
(SATISFIES A_{UNIV} OrderHDDOfHWS)

Fig. 11. Second plan at time step t_k.

The costs for the P_{k+1} are the following: $c(A_{MM}, P_{k+1})=21$, $c(A_{HWS}, P_{k+1})=24$, and $c(A_{UNIV}, P_{k+1})=15+20=35>30$. As far as A_{UNIV} is not satisfied with P_{k+1}, the replanning is performed.

The resulting plan P_{k+2} is illustrated in Fig. 12.

The costs for the P_{k+2} are the following: $c(A_{MM}, P_{k+2})=21$, $c(A_{HWS}, P_{k+2})=10+2+1+15+1=29$, $c(A_{UNIV}, P_{k+2})=15+15=30$. As far as all $c(., P_{k+2}) \leq 30$, the reconfiguration plan P_{k+2} is adopted. The assignment structure is revised, and redesigned as depicted in Fig. 13.

(SATISFIES A_{MM} OrderMonitorOfMM)
(OR_DECOMPOSES A_{HWS}, OrderKeyboard
OrderKeyboardOfMM OrderKeyboardOfHWS)
(PASSES A_{HWS} A_{MM} OrderKeyboardOfMM)
(SATISFIES A_{MM} OrderKeyboardOfMM)
(SATISFIES A_{HWS} OrderHeadphonesOfHWS)
(SATISFIES A_{HWS} OrderHDDOfHWS)
(SATISFIES A_{UNIV} GoalFromA_{MM})
(PASSES A_{HWS} A_{UNIV} OrderSpeakersOfHWS)
(SATISFIES A_{UNIV} OrderSpeakersOfHWS)

Fig. 12. Third plan at time step t_k.

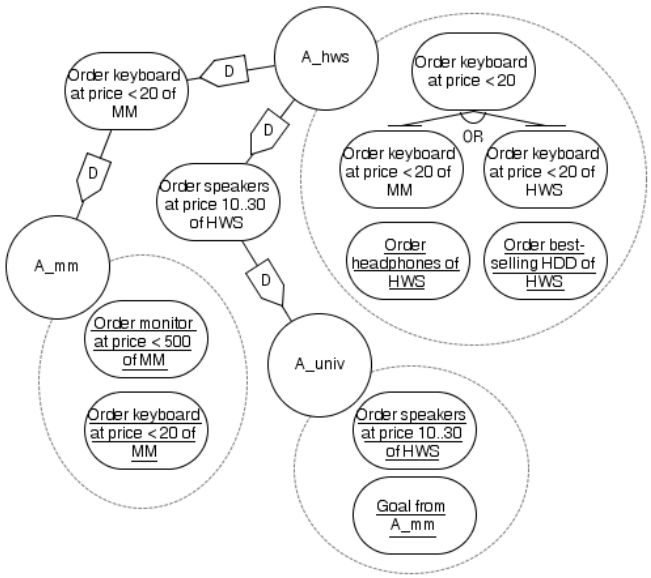

Fig. 13. Second reconfiguration of search-and-order MAS.

4. General Architecture for Self-Configuring Systems

To implement the presented approach, i.e. to add to a socio-technical system the ability to self-configure, two-layered multi-agent architecture is proposed, which is presented in Fig. 14.

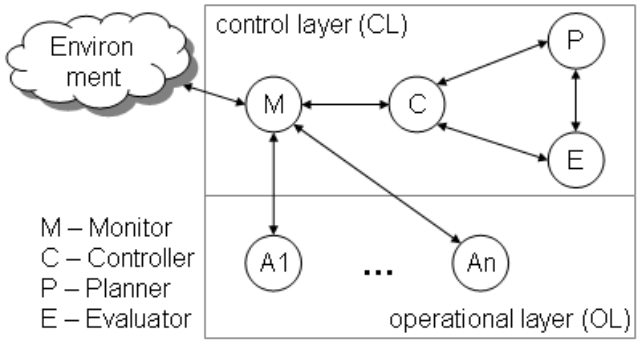

Fig. 14. Agents of the control layer.

The lower layer, called the operational layer (OL), is domain-specific, and comprises a set of agents aiming to satisfy the goal of an STS (place the orders to the suppliers, book the plane tickets, manage meeting agenda, etc.). On the upper layer, called the control layer (CL), there sit four agents – *Monitor*, *Controller*, *Planner* and *Evaluator*. *Monitor* is responsible for the communication with the agents of the operational layer, and the environment. The OL agents notify the *Monitor* about the relevant changes. *Controller*, *Planner* and *Evaluator* realize the domain-independent procedures: reconfiguration, planning and evaluation, respectively. The data they store and process (status of system goals, formal definition of a planning problem, costs of actions for each actor) is specific to a given STS. *Controller* is following the reconfiguration mechanism presented in Section 3.1, delegating planning and evaluation tasks to *Planner* and *Evaluator*, respectively. The new system configuration, produced by *Controller*, *Planner* and *Evaluator*, is propagated to the OL agents by *Monitor*. The separation of duties between the control layer agents is detailed in Table 2.

Table 2. Agents of the control layer.

Agent	Data Stored	Actions Performed	Communication
Monitor	Read-only access to *problemDef*, *D*, *G*.	Monitors (listens) OL and the environment; notifies *Controller* about triggering changes; propagates the new plan to OL agents.	Environment, OL agents, *Controller.*
Controller	*problemDef, actionLog*, *removeActionLog*, *G*, *D*.	Follows the reconfiguration mechanism, exploiting *Planner* (to initiate replanning) and *Evaluator* (to evaluate loads); updates stored data structures; notifies *Monitor* about plan changes.	*Monitor*, *Planner*, *Evaluator.*
Planner	Domain definition, read-only access to *problemDef.*	Performs planning; tunes *problemDef* in accordance with *Evaluator's* results.	*Controller*, *Evaluator.*
Evaluator	Action costs and load bounds for each OL agent.	Follows the evaluation procedure; evaluates OL agents load.	*Controller*, *Planner.*

The authors propose that the described multi-agent architecture is to be implemented in JADE (Java Agent DEvelopment framework) [1], FIPA-compliant [2] framework for multi-agent systems development. Four agents of the control layer will have the same functionality for any domain-specific instance of the architecture. The *Controller* agent will implement the reconfiguration algorithm presented in Figure 5. The *Monitor* needs to implement the communication with OL agents and the environment (e.g. using standard FIPA protocols, like ContractNetProtocol). The functionality of the *Planner* and *Evaluator* agents have been already implemented in P-Tool, see [6] for the brief description. P-Tool is an implemented prototype to support the designer/requirements engineer in the process of exploring and evaluating design alternatives. The tool has a graphical interface for the input of actors, goals and their properties. LPG-td planner is built in the tool, and is used to generate requirements alternatives, and represents each solution graphically using *i**/Tropos notation [21, 5].

5. Conclusions

In this paper an approach to the problem of dynamic reconfiguration of socio-technical information systems in response to the internal and environmental changes has been proposed. The procedure for exploring and evaluating alternative system configurations has been described, which is based on AI planning techniques and game theoretic notions of an equilibrium and local strategies. Also the reconfiguration mechanism has been presented, which is based on the above planning-and-evaluation procedure. All steps of the approach were illustrated with a simple but illustrative example. Finally, the multi-agent architecture of a self-configuring system was discussed, and it has been shown how the approach can be implemented on the base of the presented algorithm, and the previous work of the authors on the automatic exploration of design alternatives.

The presented approach can be applied both to socio-technical systems and to multi-agent systems, which comprise only software agents. However, the application of the approach to the former type of systems can be much more beneficial, as dynamicity and the self-interested rational behavior are among the STS intrinsic properties.

The proposed reconfiguration mechanism is limited in that it supports only four types of triggering events, namely, the situations when a new actor enters the system, or the existing one leaves, when a new system goal is introduced, or one of the old ones is satisfied. However, the formalization could be quite easily extended to support the changes in the actors' capabilities and commitments, failures when achieving goals, etc. This is possible due to the flexibility of the PDDL representation [11] of the problem and the planning domain. This issue is planned to be addressed in the future work. Among the other future work directions are providing the tool support for the approach, and its verification with the help of real-life case studies.

Acknowledgements

This work has been partially funded by EU Commission, through the SENSORIA and SERENITY projects, by the FIRB program of MIUR under the ASTRO project, and also by the Provincial Authority of Trentino, through the MOSTRO project. The authors also thank anonymous reviewers of the paper for their valuable comments.

References

[1] JADE: Java Agent DEvelopment Framework website http://jade.tilab.com/.

[2] FIPA: Foundation for Intelligent Physical Agents http://www.fipa.org/.

[3] N Arshad, D Heimbigner, and A L Wolf, A planning based approach to failure recovery in distributed systems. In Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems, New York, NY, USA, 2004. ACM Press, pp. 8-12.

[4] C Bernon, M P Gleizes, S Peyruqueou, and G Picard, Adelfe: A methodology for adaptive multi-agent systems engineering. In Proceedings of ESAW'02, 2002, pp. 156-169.

[5] P Bresciani, P Giorgini, F Giunchiglia, J Mylopoulos, and A Perini, Tropos: an agent-oriented software development methodology. JAAMAS, Vol. 8, No. 3, 2004, pp. 203-236.

[6] V Bryl, P Giorgini, and J Mylopoulos, Requirements analysis for socio-technical systems: exploring and evaluating alternatives. Technical Report DIT-06-006, University of Trento, 2006.

[7] V Bryl, F Massacci, J Mylopoulos, and N Zannone, Designing security requirements models through planning. In Proceedings of CAiSE'06, 2006, pp. 33-47.

[8] L Cernuzzi and F Zambonelli, Dealing with adaptive multi-agent organizations in the gaia methodology. In Proceedings AOSE'05, 2005, pp. 217-228.

[9] S A DeLoach and E Matson, An organizational model for designing adaptive multiagent systems. In Proceedings of AAAI'04 Workshop on Agent Organizations, 2004, pp. 66-73.

[10] V Dignum, L Sonenberg, and F Dignum, Towards dynamic reorganization of agent societies. In Proceedings of Workshop on Coordination in Emergent Agent Societies, 2004.

[11] S Edelkamp and J Hoffmann, Pddl2.2: the language for the classical part of the 4th international planning competition. Technical Report 195, University of Freiburg, 2004.

[12] J F Hübner, J S Sichman, and O Boissier, Using the Moise+ for a cooperative framework of MAS reorganisation. In Proceedings of SBIA'04, 2004, pp. 506-515.

[13] LPG Homepage. LPG-td Planner. http://zeus.ing.unibs.it/lpg/.

[14]] M J Osborne and A Rubinstein, A course in game theory. MIT Press, 1994.

[15] J Peer, Web Service Composition as AI planning – a survey. Technical report, University of St. Gallen, 2005.

[16] I Sommerville, Software engineering (7th ed.). Addison-Wesley, 2004.

[17] R Sterritt, C Rouff, J L Rash, W Truszkowski, and M G Hinchey, Self*- properties in Nasa mission. In Software Engineering Research and Practice, 2005, pp. 66-72.

[18] R Turner and E Turner, A two-level, protocol-based approach to controlling autonomous oceanographic

sampling networks. IEEE Journal of Oceanic Engineering, Vol. 26, No. 4, 2001, pp. 654-666.
[19] D S Weld, Recent advances in AI planning. AI Magazine, Vol. 20, No. 2, 1999, pp. 93-123.
[20] T D Wolf and T Holvoet, Towards a methodology for engineering self-organising emergent systems. Self-Organization and Autonomic Informatics (I), Vol. 135, No. 1, 2005, pp. 18-34.
[21] E S-K Yu, Modelling strategic relationships for process reengineering. PhD thesis, University of Toronto, 1996.

Author Bios

Volha Bryl is currently a PhD student at the ICT Doctorate School in Information and Communication Technologies at the University of Trento, Italy. She received the 5-year-degree in Applied Mathematics and Computer Science from the Belarusian State University (Minsk, Belarus) in 2003. Her research interests lie in the area of multi-agent systems and agent-oriented software engineering; in particular, she works with the goal-oriented requirements analysis and design in the light of Tropos, an agent-based oriented software engineering methodology. She was also been involved in the development of ToothAgent, a multi-agent architecture aimed at supporting virtual communities of co-located users.

Paolo Giorgini is a researcher at the University of Trento. He received his PhD degree from the Computer Science Institute of the University of Ancona (Italy) in 1998. After, that he joined the University of Trento as a pos-doc researcher. In December 1998 he was a visiting researcher at the Computer Science Department of the University of Toronto (Canada), and more recently he was a visiting researcher at the Software Engineering Department of the University of Technology in Sydney. He has worked on the development of requirements and design languages for agent-based systems, and the application of knowledge representation techniques to software repositories and software development. He is one of the founders of Tropos, an agent-based oriented software engineering methodology. His publication list includes more than 130 refereed journal and conference proceedings papers and eight edited books. He has contributed to the organization of several international conferences as a chair and a program committee member, and he is Co-editor in the Chief of the International Journal of Agent-Oriented Software Engineering (IJAOSE).